Student: Soham Sahasrabuddhe **Roll Number: 23B1848**

Project Title:

Arduino-Powered RGB Matrix: From Shapes to Minesweeper Adventures

Project abstract:

1. **What's the grand goal?**
The goal of this project is to showcase the potential of Arduino to seamlessly control an RGB LED matrix through means such as hand gestures tracked via a webcam or joystick navigation, highlighting its interactive capabilities. The project features a finger movement tracer, utilizing a webcam to create dynamic, random shapes(highly discrete) on the LED matrix, alongside an engaging Minesweeper game navigable through joystick controls.

2. **What are the main inputs and outputs? i.e. during the demo, what do you expect to demonstrate?**
Inputs:
   - Hand and finger movements detected via a webcam
   - Joystick movements for interaction and control

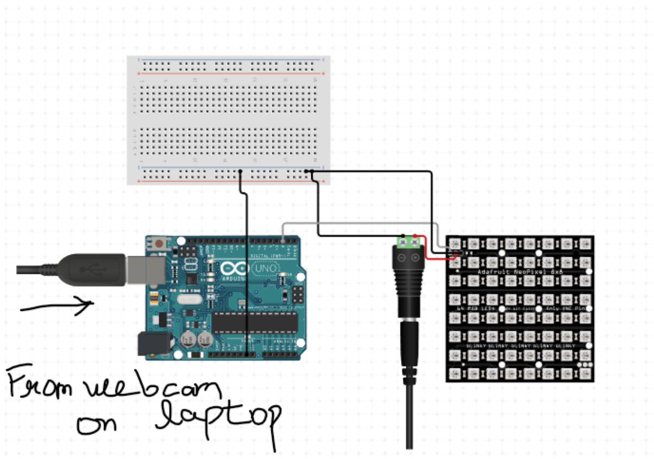
   Outputs:
   - Corresponding shapes displayed on an LED matrix, derived from the webcam's captured input
   - Minesweeper game on LED, with mines spread over the matrix

3. **What is the role of the Arduino in your project?**
In Webcam Input:
   - The Arduino receives real-time coordinates from the webcam as the user's finger moves. It parses these coordinates and maps them to the corresponding LEDs in the matrix, controlling their on/off states
   - Furthermore, it processes input to determine which hand is being used and assigns the appropriate color to the LEDs

   In Joystick input:
   - The Arduino operates on a 5V DC voltage and processes the joystick signals, including the cursor's location and the switch input, to determine actions in the Minesweeper game-—such as unveiling specific tiles or identifying mines
   - It randomly decides the mine locations with an approximate probability of 0.2 for each run, ensuring unpredictability in the game

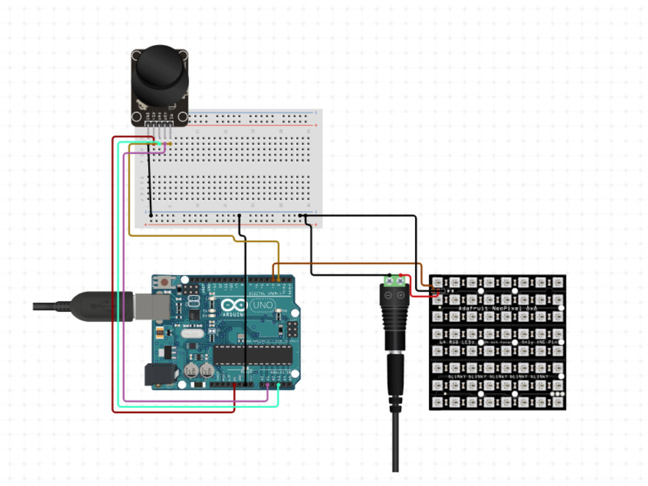
Project Detail:

Finger Tracker Circuit



The laptop will serve as both the input provider and power supply for this setup. The connection will be established via USB, enabling the laptop to transmit real-time data to the Arduino while simultaneously supplying the necessary power to operate the circuit.

Minesweeper Circuit



A 5V DC power supply, provided as part of the lab setup, will be utilized in this section.

Photos & Video

Follow this link to visit drive storing photos and videos of my project!

Sensors and output devices

The sensor in this setup includes the webcam integrated within the laptop, which is used for the initial functionality. For the second part of the project, a joystick serves as an additional sensor for enhanced interactivity.
The output device is an 8x8 Adafruit RGB LED Matrix, responsible for visually rendering the processed data or results based on the input received from the sensors.

Pre-installed Libraries

List of libraries I used are:

1. **OpenCV**: Library for computer vision tasks, used for image and video processing, object detection, and tracking. It facilitates webcam-based tracking in my project.

2. **Mediapipe**: Framework by Google for building cross-platform ML solutions. It's especially useful for hand and face tracking, which can be integrated with OpenCV used above for advanced detection.

3. **Serial**: Python library used for serial communication between devices like a laptop and Arduino

4. **Adafruit_Neopixel**: Library for controlling RGB LED matrices and strips, used to drive LED in my project

Milestones:

1. **Week 1:** Learnt what is OpenCV, basic commands and functions, how to use it in my context, what are necessary modules to be imported, and researched about how to give real time inputs to Arduino using USB (serial communication)

2. **Week 2:** Started building the tracer physical circuit for the tracer part, using a 8x8 cathode LED matrix, checked whether each led was working or not, figured out the pin configuration, and connected it to LED driving resistors and Arduino. Also started writing the python code for the webcam tracking, based on modules learnt last week

3. **Week 3:** Wrote the Arduino part of the logic, to take inputs from the laptop process the coordinates, and other secondary data to control the LED matrix, turning ON & OFF. Replaced the current physical matrix with RGB one as its intensity was not enough and some of the LEDs were faulty. Figured out the RGB code, and its library Adafruit.Neopixel, python logic being the same. Made my first demo in front of the TA

4. **Week 4:** Started testing edge cases, added grid to the webcam for better tracking of finger based on reviews of friends. Also added green colour, if the grid is selected for memory purposes, added an option of using two hands for drawing, bringing two colours on LED, leveraging the fact that I was using a RGB Matrix. This also inspired me to create a minesweeper game. Ideated the whole logic and game rules to be put in on paper, and took a joystick from friend as he had one extra, to control the cursor movement.

5. **Week 5:** Coded the whole minesweeper logic into Arduino code, tested out edge cases, added a functionality of cursor and press option to select tiles to unveil. Made the joystick circuit with Arduino and tried it out with 5 V DC supply

Program Code:

Python: For webcam inputs

```
import cv2
import mediapipe as mp
import serial
import time
import numpy as np

# Initialize Serial Communication using USB to send input
arduino = serial.Serial(port="COM5", baudrate=9600, timeout=1)
time.sleep(2)

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=2,
   ↪ min_detection_confidence=0.5)
mp_drawing = mp.solutions.drawing_utils

# Webcam
cap = cv2.VideoCapture(0)
grid_state = np.zeros((8, 8), dtype=int)  # Only track active cells
hand_labels = {}

def map_to_grid(x, y):
    x_grid = min(max(int(x * 8), 0), 7)
    y_grid = min(max(int(y * 8), 0), 7)
    return x_grid, y_grid

def draw_grid_overlay(frame, grid_state):
    h, w, _ = frame.shape
    cell_size = min(h, w) // 8

    start_x = (w - (cell_size * 8)) // 2
    start_y = (h - (cell_size * 8)) // 2

    for i in range(8):
        for j in range(8):
            top_left = (start_x + j * cell_size, start_y + i * cell_size)
            bottom_right = (start_x + (j + 1) * cell_size, start_y + (i + 1)
               ↪ * cell_size)

            # Activated Cell
            if grid_state[i][j]:
                cv2.rectangle(frame, top_left, bottom_right, (0, 255, 0), -1)

            # White grid lines
            cv2.rectangle(frame, top_left, bottom_right, (255, 255, 255), 1)

while cap.isOpened():
    ret, frame = cap.read()
    frame = cv2.flip(frame, 1)
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(rgb_frame)

    if results.multi_hand_landmarks and results.multi_handedness:
        for i, hand_landmarks in enumerate(results.multi_hand_landmarks):
            mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.
               ↪ HAND_CONNECTIONS)
            hand_label = results.multi_handedness[i].classification[0].label
               ↪  # Left or Right Hand
            hand_labels[hand_label] = hand_landmarks

        # Index Fingers
        for hand, label in hand_labels.items():
            index_tip = label.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP
               ↪ ]
            x_grid, y_grid = map_to_grid(index_tip.x, index_tip.y)

            if grid_state[y_grid][x_grid] == 0:  # Activate cell if not
               ↪ already active
                grid_state[y_grid][x_grid] = 1
                arduino.write(f"{x_grid},{y_grid},{hand}\n".encode())

    draw_grid_overlay(frame, grid_state)
    cv2.imshow("Finger Drawing", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
arduino.close()
```

Arduino: For tracing the shape drawn

```
#include <Adafruit_NeoPixel.h>

// Pin definition
#define LED_PIN 2
#define MATRIX_SIZE 64
#define BRIGHTNESS 50

Adafruit_NeoPixel matrix = Adafruit_NeoPixel(MATRIX_SIZE, LED_PIN, NEO_GRB +
   ↪ NEO_KHZ800);

void setup() {
    Serial.begin(9600);
    matrix.begin();
    matrix.setBrightness(BRIGHTNESS);
    matrix.show();}

void loop() {
    if (Serial.available() > 0) {
        String receivedData = Serial.readStringUntil('\n');
        int x, y;
        char hand[10];
        // Parsing the received data
        sscanf(receivedData.c_str(), "%d,%d,%s", &x, &y, hand);

        if (x >= 0 && x < 8 && y >= 0 && y < 8) {
            int ledIndex = y * 8 + x;  // Convert (x,y) to single LED index

            // Determine color based on hand
            if (strcmp(hand, "Left") == 0) {
                matrix.setPixelColor(ledIndex, matrix.Color(255, 0, 0));} //
                   ↪ Red for Left hand
            else if (strcmp(hand, "Right") == 0) {
                matrix.setPixelColor(ledIndex, matrix.Color(0, 0, 255));} //
                    ↪ Blue for Right hand
            matrix.show(); } // Refreshing the LED
    }
}
```

Arduino: Minesweeper

```
#include <Adafruit_NeoPixel.h>

// Pin Definitions
#define LED_PIN 2
#define NUM_LEDS 64
#define JOY_X A0
#define JOY_Y A1
#define JOY_SW 12
#define BRIGHTNESS 40

Adafruit_NeoPixel matrix(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800); //
    ↪ Initialising the matrix

int cursorX = 0, cursorY = 7;
int grid[8][8];
bool revealed[8][8] = {false};
bool buttonPressed = false;

void setup() {
    Serial.begin(9600);
    matrix.begin();
    matrix.clear();
    matrix.setBrightness(BRIGHTNESS);
    pinMode(JOY_SW, INPUT_PULLUP);
    randomSeed(analogRead(0));
    generateMinesweeperGrid();
    updateLEDs();}

void loop() {
    handleJoystick();
    if (digitalRead(JOY_SW) == LOW) {
        if (!buttonPressed) {
            revealTile(cursorX, cursorY);
            updateLEDs();
            buttonPressed = true;}}
    else {
        buttonPressed = false;}
    delay(100);}

void generateMinesweeperGrid() {
    // First clear the grid
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            grid[i][j] = -1; // no mine
        }}

    // Place mines
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (random(10) < 2) {
                grid[i][j] = 0;} // mine
        }}

    // Calculate adjacent mines for all non-mine tiles
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (grid[i][j] != 0) {
                int mines = countAdjacentMines(i, j);
                grid[i][j] = mines;
                if (mines == 0) {
                    grid[i][j] = 5;}
            }}}}

int countAdjacentMines(int x, int y) {
    int count = 0;
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            if (dx == 0 && dy == 0) continue;
            int nx = x + dx, ny = y + dy;
            if (nx >= 0 && nx < 8 && ny >= 0 && ny < 8 && grid[nx][ny] == 0)
               ↪ {
                count++;}
        }}
    return count;}

// Once clicked, reveal the tile
void revealTile(int x, int y) {
    if (x < 0 || x >= 8 || y < 0 || y >= 8 || revealed[x][y]) return;
    revealed[x][y] = true;
    if (grid[x][y] == 5) {
        for (int dx = -1; dx <= 1; dx++) {
            for (int dy = -1; dy <= 1; dy++) {
                if (dx == 0 && dy == 0) continue;
                int nx = x + dx, ny = y + dy;
                if (nx >= 0 && nx < 8 && ny >= 0 && ny < 8 && grid[nx][ny] !=
                   ↪  0) {
                    revealTile(nx, ny);}
            }}}}

void updateLEDs() {
    matrix.clear();
    // Draw revealed tiles with proper colors
    for (int x = 0; x < 8; x++) {
        for (int y = 0; y < 8; y++) {
            if (revealed[x][y]) {
                setLEDColor(x, y, getColor(grid[x][y]));}}}
    // Draw cursor (white)
    setLEDColor(cursorX, cursorY, matrix.Color(255, 255, 255));
    matrix.show();}

// Game rules and logic behind the tile colours
uint32_t getColor(int value) {
    switch (value) {
        case 0:  return matrix.Color(255, 0, 0);      // Mine - Red
        case 1:  return matrix.Color(255, 200, 0);    // 1 adjacent mine -
           ↪ Yellow
        case 2:  return matrix.Color(0, 0, 255);      // 2 adjacent mines -
           ↪ Blue
        case 3:  return matrix.Color(150, 0, 150);    // 3 adjacent mines -
           ↪ Violet
        case 4:  return matrix.Color(0, 120, 200);  // 4 adjacent mines -
           ↪ Light Blue
        case 5:  return matrix.Color(0, 255, 0);      // Safe (no adjacent
           ↪ mines) - Green
        default: return matrix.Color(0, 0, 0);        // Shouldn't happen
    }}

void setLEDColor(int x, int y, uint32_t color) {
    int index = (7 - y) * 8 + x;
    matrix.setPixelColor(index, color);}

// Joystick function to control cursor movement
void handleJoystick() {
    int xValue = analogRead(JOY_X);
    int yValue = analogRead(JOY_Y);
    if (xValue < 400) cursorX = max(0, cursorX - 1);
    else if (xValue > 600) cursorX = min(7, cursorX + 1);
    if (yValue < 400) cursorY = min(7, cursorY + 1);
    else if (yValue > 600) cursorY = max(0, cursorY - 1);
    updateLEDs();
    delay(100);
}
```